judge.openfmi.net:9080/mediawiki/index.php/Максимален\_поток 29.01.2017, 03:04 ч.

03.06.2018, 03:22 ч,

04.06.2018, 12:32 ч.

По-добър алгоритъм

ще покажа и един по-добър алгоритъм за намиране на max-flow.

Може да го намерите в Интернет под името **Dinits**.

Алгоритъмът се състои в следното:  
Означения:

**sourse** – изходен връх

**sink** – краен връх (цел, получател, target)

**Gf** – граф на потоците (graph flow)

**1**) **пускаме bfs от sink** по обратните ребра (за да стигне от i до j, трябва в Gf стойността на реброто от j до i да е ненулева). **По този начин за всеки връх изчисляваме нивото му спрямо sink** (дължината на най-краткото му разстояние до sink).   
**Ако при това обхождане не посетим source**, няма път в Gf от source до sink и алгоритъмът приключва

**2) Пускаме dfs от source** по нормалните ребра, като се движим само по тези, с които се намалява разстоянието до sink, т.е. движим се по нивата, намерени с bfs. (за да стигне от i до j, трябва Gf стойността на реброто от i до j да е ненулева)  
**2.1**) **Ако стигнем до sink:** правим source и sink неизползвани (used[sink] = used[source] = 0), намираме най 'тясното' ребро по намерения път, напр. с име CurrFlow. Увеличаваме потока с CurrFlow по намерения път от source до sink (отразяваме промените и в графа Gf). Повтаряме от точка 2  
**2.2**) **Ако не стигнем до sink:** правим непосетени всички върхове. Повтаряме от точка 1.  
  
Интересното при този алгоритъм е, че сложността на dfs-а (точка 2) между 2 пускания на bfs (точка 1) е колкото сложността на 1 dfs. Това е така, защото всеки връх (освен source и sink) се обхожда максимум веднъж ( защото ако 1 dfs направи даден връх използван (used[a] = 1), то следващото dfs няма да мине през него). Забележете че 'изчистването' на масива used, не се прави след всяко dfs.  
Сложността на този алгоритъм за matching е O( sqrt(n).m)

Разширяване на проблема

Разгледаният по горе проблем се занимава само с един източник(source - t) и едно място където отиват стоките(sink - t), но това лесно може да се промени.

Добавя се SuperSource и SuperSink и се разширява графа така че от тях да излизат/влизат ребра към/от останалите източници/складове.

Материали

Статията е подготвена основно по материали от "Introduction to Algorithms" 2ed. MIT Press и от <http://homepages.cwi.nl/~lex/files/agtco.flows.pdf>. За повече материали и за подробни и доста разбираеми доказателства на по горните твърдения погледнете там(особено в Intro-то).

<http://e-maxx.ru/algo/Dinitz> Диригенският алгоритъм

Формулиране на проблема

## Постановка на задачата

Нека е дадена мрежа, т.е. ориентиран граф G, в който към всяко ребро (u, v) е присвоен капацитет(пропускателна способност, широчина на лентата) cuv и два върха - източник s и краен връх t.

Да се намери в тази мрежа максималният поток fuv от източника s до t

## Малко история

Този алгоритъм е публикуван от съветския (израелския) учен Efim Dinitz през 1970 г., т.е. дори две години преди публикуването на алгоритъма Edmonds-Carp (въпреки че и двата алгоритъма са открити независимо през 1968 г.).

В допълнение, трябва да се отбележи, че някои опростявания на алгоритъма са направени от Simon Dori (Шимон Дори) и неговия ученик Alon Itai (Алън Итаи) през 1979 г. Благодарение на тях алгоритъмът е получил сегашния си вид - те използват идеята на концепцията за блокиране на потоци на Diniz Alexander Karzanov (Александър Казанов, 1974), а също и преформулират алгоритъма за тази комбинация на обхождане по ширина и дълбочина, в който вид този алгоритъм е описан навсякъде.

Разработването на идеи във връзка с алгоритмите за потоци е изключително интересно, като се има предвид "Желязната завеса" от онези години, разделящи СССР и Запада. Очевидно е, както е понякога Подобни идеи показват почти едновременно (както в случая на алгоритъма и алгоритъм Diniz Edmond-Karp), обаче, като същевременно различна ефективност (Dinitz е един порядък по-бърз);

Понякога, наобратно, появата на идеята от една страна на "желязната завеса" е подобен ход, a от друга страна се проточва в продължение на повече от десет години (както алгоритъма на Karzanov - през 1974 г., и алгоритъма на Goldberg(Голдбърг) юпрез 1985 г.).

## Необходими определения

Въвеждаме три необходими дефиниции (всяка от тях е независима от другите), които след това ще бъдат използвани в алгоритъма на Dinitz.

**Остатъчна мрежа GR** по отношение на мрежата GF и определен поток в нея се нарича мрежа, в която всяко ребро (u,v) в G с капацитет Cuv и поток fuv съответства на две ребра:

(u, v) с капацитет cuvR = cuv - fuv

(v, u) с капацитет cvuR = fuv

Трябва да се отбележи, че с тази дефиниция в остатъчната мрежа могат да се появят множество ребра (u, v) и (v, u), ако в първоначалната мрежа има цикъл

**Остатъчно ребро** може интуитивно да се разбира като мярка за това колко повече може да се увеличи потокът по някакво ребро. В действителност, ако по ребро (u,v) с капацитет Cuv преминава поток fuv,

то потенциалът в него може да пропусне още Cuv-fuv единици поток и в обратна посока е възможно да се пропусне поток до fuv, което би означавало анулиране(отмяна) на потока в първоначалната посока.

**Блокиращ поток** в дадена мрежа е поток, така че всеки път от източника s към t съдържа ребро, наситено с този поток. С други думи, в тази мрежа няма начин, по който да се увеличи без затруднения потокът от източника до t.

**Блокиращият поток не е непременно максимален.** Теоремата на Форд-Фулкерсон гласи, че **потокът ще бъде максимален, тогава и само тогава, когато няма пътища s-t в остатъчната мрежа**;

В блокиращата мрежа нищо не се казва за съществуването на път по ребрата, появяващи се в остатъчната мрежа.

Многопластовата (многослойна, слоеста, слоиста) мрежа за дадената мрежа се изгражда по следния начин.

**Първо се определят дължините на най-късите пътища от източника s до всички други върхове;**

Ще наричаме с level[v] (ниво на върха) неговото разстояние от източника.

**След това многослойната мрежа включва всички ребра (u, v) на дадената мрежа, които водят от едно ниво до друго, по-голямо ниво**, т.е. level[u] + 1 = level[v] (това, че разликата в разстоянието не може да надвишава 1, следва от свойството за най-кратките разстояния).

**По този начин се намират всички ребра, които са изцяло в нивото, както и ребрата, водещи към предишните нива.**

Очевидно е, че мнослойната мрежа е ациклична. Освен това, всеки път от s до t в многослойна мрежа е най-краткият път в мрежата на източника.

**Изграждането на слоеста(многослойна) мрежа на дадената мрежа е много лесно:** за целта трябва да се пусне обхождае в ширина по ребрате на мрежата, като се изичслява всяка стойност за връх с ниво level[], и след това към слоестата мрежа се добавят всички подходящи ребра.

Забележка. Терминът "слоеста мрежа" не се използва в руската езикова литература; Обикновено тази конструкция се нарича просто "помощен граф". Въпреки това, на английски терминът "layered network" ("слоеста мрежа") често се използва.

## Схема на алгоритъма

Алгоритъмът се състои ит няколко фази.

На всеки етап отначало се строи остатъчната мрежа, след което се към нея се конструира слоеста(по нива) мрежа (обхождайки по ширина) и се търси произволен блокиращ поток. Откритият блокиращ поток се добавя към текущия поток и това завършва следващата итерация.

Този алгоритъм е подобен на алгоритъма на Едмондс-Karp, но основната разлика е, че тук при всяка итерация потокът се увеличава не по протежение на един най-кратък път S-т, а по целия набор от начини (именно такива са пътищата на блокиращия поток в многослойната мрежа

===

## Коректност на алгоритъма

Ще покажем, че когато алгоритъмът завърш, изходът от него е потокът от максималната стойност.

Нека да предположим, че в даден момент в многослойната мрежа, изградена за остатъчната мрежа, не е било възможно да се намери блокиращ поток. Това означава, че t не е достижимо от източника s.

Но тъй като многослойната мрежа съдържа всички най-кратки пътища от източника в остатъчната мрежа, това на свой ред означава, че няма път от източника към изтичането в остатъчната мрежа. Следователно, прилагайки теоремата на Форд-Фолкерсън, получаваме, че текущият поток всъщност е максимален.

## Оценка на броя на фазите

Ще покажем, че алгитимът на Dinitz винаги изпълнява по-малко от n фази.

За тази цел ще докажем две леми:

**ЛЕМА 1.** Най-късото разстояние от източника до всеки връх не намалява с изпълнението на всяка итерация, т.е.

 leveli + 1[v] ≥ leveli [v]

където индексът означава номера на фазата, преди която са взети стойностите на тези променливи.

Доказателство. Фиксираме произволна фаза i и произволен връх V и разглеждаме най-краткия S-V път P в мрежата GRi+1 (Спомнете си, че така означаваме остатъчната мрежа, получена преди изпълнението на (i+1)-та фаза.) Очевидно дължината на пътя P е leveli + 1[v].

Отбелязваме, че в остатъчната мрежа GRi+1 могат да се съдържат само ребра от GR, а такива ребра са обратни (това следва от определението за остатъчна мрежа).

Нека разгледаме два случая:

Път P съдържа само ребра в GR. Тогава, е разбираемо, че дължина на пътя Р е по-голяма от или равна на leveli [V] (защото leveli [V] по дефиниция е дължината на най-късия път), което означава, че неравенството е изпълнено.

Пътят P съдържа поне едно ребро, което не се съдържа в GR (но е обратно на някои ребра в GR). Нека пъррвото такова ребро е (u, w).

s ===> u --> ===> v

 Можем да приложим нашата лема за върха u, защото тя попада в първия случай; Така че получаваме неравенството leveli + 1[u] ≥ leveli [u]

Сега ще отбележим, че тъй като реброто (u.w) се появава в остатъчната мрежа само след i-тата фаза, от това следва, че по реброто (w, u) е бил пропуснат допълнително някой поток; Следователно, реброто (w, u) е принадлежало на многослойната мрежа преди i-тата фаза, а после leveli[u]=leveli[w]+1.

Като отчетем свойството за кратките пътища leveli + 1[w] = leveli + 1[u]+1 и комбинираме това равенство с предишните две неравенства, получаваме:

leveli + 1[w] ≥ leveli [w] + 2

Сега можем да приложим същите разсъждения за целия оставащ път към V (т.е., че всяко обратно ребро добавя към нивото най-малко две), и като резултат ще получим необходимата неравенство.

**ЛЕМА 2.** Разстоянието между източника и края се увеличава стриктно след всяка фаза на алгоритъма, т.е .:

 level’[t] > level[t]

където с апостроф е указана стойността, получена в следващата фаза на алгоритъма.

**Доказателство**: чрез противоречие. Да предположим, че след текущата фаза се оказа, че  level’[t] = level[t]

Разглеждаме най-краткия път от източника до края; с предположение, че дължината му остава непроменена. Но остатъчната мрежа в следващата фаза съдържа само ребрата на остатъчната мрежа преди изпълнението на текущата фаза или обратните им стойности. По този начин стигнахме до противоречие: немерен е път s-t, който не съдържа наситени ребра и има същата дължина, като най-краткия път. Този път трябваше да бъде "блокиран" от блокиращия поток, което не се е случило и това е противоречието, което трябваше да бъде доказано.

Тази лема може да бъде интуитивно разбрана, както следва: на i-тата фаза, алгоритъмът на Dinitz идентифицира и насища всички s-t пътища с дължина i.

Тъй като дължината на най-късия път от s до t не може да надвишава n-1, следва, че алгитимът Dinitz не надхвърля n-1 фази.

## Търсене на блокиращ поток

За да завършим изграждането на алгоритъма на Dinitz, трябва да опишем алгоритъма за намиране на блокиращия поток в слоеста мрежа - ключовото място на алгоритъма.

Ще разгледаме три възможни начина за изпълнение на търсенето на блокиращ поток:

* **Начин 1:** Търсим пътищата s-t един по един, докато такива пътища се намират. Пътят може да се намери за O (m) обхождане в дълбочина, а всички такива пътища ще бъдат O (m) (тъй като всеки път насища поне едно ребро). Получената асимптотика на търсенето на един блокиращ поток ще бъде 0 (m2).
* **Начин 2:** Подобно на предишната идея, но в процеса на обхождане по дълбочина, изтриваме от графа всички "излишни" ребра, т.е. ребра, по които не е възможно да се стигне до t.

**Това много лесно може да е реализира:** просто премахаме ребро, след като сме пуснали обхождане в дълбочина от него (с изключение на случаите, когато сме минали по реброто и сме намерили път до t).

От гледна точка на реализацията, е необходимо просто да се поддържа в списъка съседи за всеки връх указател към първото неизтрито ребро и да се увеличава в цилъл във вътрешнпстта при обхождането в дълбочина.

**Да оценим асимптотиката на това решение.** Всяко обхождане в дълбочина завършва или с насищане на най-малко едно ребро (ако при това обхождане се стигне до t), или с предвижване напред най-малко с един указател (в противен случай). Може да се разбере, че едно пускане на обхождане в дълбочина от основната програма работи за O (k + n), където k е броят на предвижване на указателите. Като се има предвид, че всички пускания на обхождане в дълбочина при търсене на един блокиращ поток е О (р), където р - брой ребра, наситен с този блокиращ поток, целият алгоритъм за търсене на блокиращ поток ще работи за О (p.k + p.n), което отчитайки, че всички указатели са преминали общо(сумарно) разстояние O (m), дава асимптотиката О (m + pn). В най-лошия случай, когато блокиращият поток насища всички ребра, асимптотиката е О (n.m); Тази асимптотика ще бъде използвана по-късно.

Може да се каже, че този метод за намиране на блокиращ поток е изключително ефективен в смисъл, че изразходва O (n) операции средно при търсене на един увеличаващ се път. Това е именно разликата в поредността на ефективността на алгоритмите Diniz и Edmonds-Karp (който търси един увеличаващ се път за O (m)).

Този метод за решаване все още е лесен за изпълнение, но е достатъчно ефективен и следователно най-често се използва на практика.

* **Начин 3:** Могат да се използват специални структури от данни - динамични дървета Sleator (Sleator) и Tarjan (Tarjan)). Тогава всеки блокиращ поток може да се намери за време O (m .log n).

## Асимптотика

Следователно, целият алгоритъм на Dinitz е O (n2. m), ако блокиращият поток се търси по описания по-горе начин за O (n.m). Прилагането с помощта на динамични дървета на Sator и Tarjan ще работи за време O (n.m.log n).

## Единични мрежи

"Единична мрежа"(“unit network”) се нарича таква мрежа, при която капацитетите на всички съществуващи ребра са равни на 1 и всеки възел, с изключение на източника и таргета, са или входящи или изходящи.

Този случай е много важен, тъй като в задачата за намиране на максимални двойкосъчетания изградената мрежа е точно единична.

**Ще докажем, че** за единичните мрежи алгоритъмът на Diniz дори в проста реализация (която за произволни графи е O (N2. m) ) работи за време O (m \sqrt{n}), достигайки до задачата за намирането на максимално двойкосъчетание на един от най-известните алгоритми - алгоритъмът на Hopcroft-Carp.

Първо ще отбележим, че алгоритъмът за търсене на блокиращ поток, който за произволна мрежа работи за време O (n.m), в мрежи с единичен капацитет ще работи за O (m), имайки предвид, че всяко ребро няма да се разглежда по повече от един път..

На второ място ще оценим общия брой фази, които биха могли да възникнат в случай на единични мрежи.

Нека вече са генерирани \sqrt{n}  фази на лагоритъма на Dinitz. Тогава всички увеличаващи се пътища с дължина не по-голяма от \sqrt{n}  вече са намерени. Нека f е текущият поток, който е намерен, и f\* е търсеният максимален поток;

Разглеждаме разликата им: f\* - f. Това е поток в остатъчната мрежа GR. Този поток има стойността | f\* | - | f | и по всяко ребро е 0 или 1.

Той може да бъде да се декомпозира( да се разгради) на колекция от | f\* | - | f | пътища(и евентуално цикли) от s до t.

Тъй като мрежата е единична, всички тези пътища не могат да имат общи върхове, поради което, като се вземе предвид горното, общият брой върхове в тях cnt може да се оцени като:

 cnt ≥ ( | f\*| - | f | ) . \sqrt{n}

От друга страна, като се има предвид, че cnt ≤ n, от това получаваме:

 | f\* | - | f | ≤ \sqrt{n}

което означава, че дори и през \sqrt{n} фазата на алгоритъма на Dinitz има гарантиран максимален поток.

Следователно, общият брой фази на алгоритъма на Dinitzа, изпълняван по единични мрежи, може да се оцени като 2\sqrt{n} което трябваше да бъде доказано.

## Реализация

Даваме две реализации на алгоритъма за O (n2 m), работещи по мрежи, дефинирани съответно от матрици на съседство и списъци на съседи.

Изпълнение с матрици на съседство:

const int MAXN = ...; // число вершин

const int INF = 1000000000; // константа-бесконечность

int n, c[MAXN][MAXN], f[MAXN][MAXN], s, t, d[MAXN], ptr[MAXN], q[MAXN];

bool bfs() {

int qh=0, qt=0;

q[qt++] = s;

memset (d, -1, n \* sizeof d[0]);

d[s] = 0;

while (qh < qt) {

int v = q[qh++];

for (int to=0; to<n; ++to)

if (d[to] == -1 && f[v][to] < c[v][to]) {

q[qt++] = to;

d[to] = d[v] + 1;

}

}

return d[t] != -1;

}

int dfs (int v, int flow) {

if (!flow) return 0;

if (v == t) return flow;

for (int & to=ptr[v]; to<n; ++to) {

if (d[to] != d[v] + 1) continue;

int pushed = dfs (to, min (flow, c[v][to] - f[v][to]));

if (pushed) {

f[v][to] += pushed;

f[to][v] -= pushed;

return pushed;

}

}

return 0;

}

int Dinitz() {

int flow = 0;

for (;;) {

if (!bfs()) break;

memset (ptr, 0, n \* sizeof ptr[0]);

while (int pushed = dfs (s, INF))

flow += pushed;

}

return flow;

}

Сеть должна быть предварительно считана: должны быть заданы переменные n, s, t, а также считана матрица пропускных способностей c[][]. Основная функция решения — \rm dinic(), которая возвращает величину найденного максимального потока.

## Реализация със списък на съседи

const int MAXN = ...; // число вершин

const int INF = 1000000000; // константа-бесконечность

struct edge {

int a, b, cap, flow;

};

int n, s, t, d[MAXN], ptr[MAXN], q[MAXN];

vector<edge> e;

vector<int> g[MAXN];

void add\_edge (int a, int b, int cap) {

edge e1 = { a, b, cap, 0 };

edge e2 = { b, a, 0, 0 };

g[a].push\_back ((int) e.size());

e.push\_back (e1);

g[b].push\_back ((int) e.size());

e.push\_back (e2);

}

bool bfs() {

int qh=0, qt=0;

q[qt++] = s;

memset (d, -1, n \* sizeof d[0]);

d[s] = 0;

while (qh < qt && d[t] == -1) {

int v = q[qh++];

for (size\_t i=0; i<g[v].size(); ++i) {

int id = g[v][i],

to = e[id].b;

if (d[to] == -1 && e[id].flow < e[id].cap) {

q[qt++] = to;

d[to] = d[v] + 1;

}

}

}

return d[t] != -1;

}

int dfs (int v, int flow) {

if (!flow) return 0;

if (v == t) return flow;

for (; ptr[v]<(int)g[v].size(); ++ptr[v]) {

int id = g[v][ptr[v]],

to = e[id].b;

if (d[to] != d[v] + 1) continue;

int pushed = dfs (to, min (flow, e[id].cap - e[id].flow));

if (pushed) {

e[id].flow += pushed;

e[id^1].flow -= pushed;

return pushed;

}

}

return 0;

}

int Dinitz() {

int flow = 0;

for (;;) {

if (!bfs()) break;

memset (ptr, 0, n \* sizeof ptr[0]);

while (int pushed = dfs (s, INF))

flow += pushed;

}

return flow;

}

Сеть должна быть предварительно считана: должны быть заданы переменные n, s, t, а также добавлены все рёбра (ориентированные) с помощью вызовов функции \rm add\_edge. Основная функция решения — \rm dinic(), которая возвращает величину найденного максимального потока.

* Аватар

**ivan** • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-745951419)

что такое массив ptr[]?

* + [2](https://disqus.com/embed/comments/?base=default&f=e-maxx&t_u=http%3A%2F%2Fe-maxx.ru%2Falgo%2Fdinic&t_d=MAXimal%20%3A%3A%20algo%20%3A%3A%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B8%D0%BD%D0%B8%D1%86%D0%B0%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0&t_t=MAXimal%20%3A%3A%20algo%20%3A%3A%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B8%D0%BD%D0%B8%D1%86%D0%B0%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0&s_o=default" \o ")

Аватар

**yoihito** [ivan](http://e-maxx.ru/algo/dinic#comment-745951419) • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-821025372)

в нем запоминаются на каком ребре остановились для вершины v.

(в него се запомня на кое ребро сме за връх v)

* + - [1](https://disqus.com/embed/comments/?base=default&f=e-maxx&t_u=http%3A%2F%2Fe-maxx.ru%2Falgo%2Fdinic&t_d=MAXimal%20%3A%3A%20algo%20%3A%3A%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B8%D0%BD%D0%B8%D1%86%D0%B0%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0&t_t=MAXimal%20%3A%3A%20algo%20%3A%3A%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B8%D0%BD%D0%B8%D1%86%D0%B0%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0&s_o=default)

Аватар

**Павел** [yoihito](http://e-maxx.ru/algo/dinic#comment-821025372) • [3 года назад](http://e-maxx.ru/algo/dinic#comment-2316002232)

нафиг он нужен можно ведь просто написать i=0...n



Аватар

**Павел** [Павел](http://e-maxx.ru/algo/dinic#comment-2316002232) • [3 года назад](http://e-maxx.ru/algo/dinic#comment-2316006293)

А, хотя нет, я не прав

Аватар

**Евгений** • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-958607152)

Что означает запись id^1

Аватар

**Ваня** [Евгений](http://e-maxx.ru/algo/dinic#comment-958607152) • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-972920670)

Это означает обратное ребро, так как ребра в e добавляются сначала прямое в четные позиции, затем обратное для в нечетные.

Так например для 2 ^ 1 будет 3, а три 3 ^ 1 будет 2. То есть если число четное, то +1, иначе -1.

А вообще это обычный xor 1.

( това означава обратно ребро, понеже се добавят отначало так: право – в четни позиции, а след това – обратното в нечетни. Например: 2 ^ 1 = 3, а 3 ^ 1 = 2, т.е. ако числото е четно, се прибавя 1,, иначе се изважда 1 , т.е. +1, иначе -1.

[](https://disqus.com/by/victorarmegioiu/)

[**Victor Armegioiu**](https://disqus.com/by/victorarmegioiu/) • [5 месяцев назад](http://e-maxx.ru/algo/dinic#comment-3680557278)

Is the adjacency list variant really correct? I tried using it for the FASTFLOW problem on SPOJ and gave a wrong answer right from the first test. The flow found by this algorithm was equal to 3 instead of 5.

Аватар

**Roman** • [6 месяцев назад](http://e-maxx.ru/algo/dinic#comment-3647966597)

Подскажите, а какое действие выполняет функция bfs и dfs&

Аватар

**Vlad** • [год назад](http://e-maxx.ru/algo/dinic#comment-3036489429)

Есть полный код с библиотеками? и примером как это работает?

[Аватар](https://disqus.com/by/temirulanmussayev/)

[**Temirulan Mussayev**](https://disqus.com/by/temirulanmussayev/) • [2 года назад](http://e-maxx.ru/algo/dinic#comment-2912281855)

В статье сказано: "Поскольку сеть единична, то все эти пути не могут иметь общих вершин."  
Почему? Пути точно не пересекаются по ребрам и аналогично Вашим рассуждениям получаем асимптотику

O(2 \* sqrt(m)).  
В чем я ошибаюсь?

[Аватар](https://disqus.com/by/vstanka/)

[**vstanka**](https://disqus.com/by/vstanka/) [Temirulan Mussayev](http://e-maxx.ru/algo/dinic#comment-2912281855) • [5 месяцев назад](http://e-maxx.ru/algo/dinic#comment-3675730708)

Ни в чём, это в статье ошибка

[Аватар](https://disqus.com/by/disqus_kIJAex3ccl/)

[**Максим Сурков**](https://disqus.com/by/disqus_kIJAex3ccl/) • [2 года назад](http://e-maxx.ru/algo/dinic#comment-2722088708)

Какой поток является блокирующим в сети :   
Таблица пропускных способностей  
c[0][1] = 4, c[1][0] = 0  
c[1][3] = 5, c[3][1] = 0  
c[0][2] = 1, c[2][0] = 0  
c[2][3] = 1, c[3][2] = 0  
s = 0  
t = 3  
?

Аватар

**Гриша** • [2 года назад](http://e-maxx.ru/algo/dinic#comment-2442956622)

А зачем вообще нужны обратные ребра

[](https://disqus.com/by/disqus_feeApt1mrO/)

[**Akash Goel**](https://disqus.com/by/disqus_feeApt1mrO/) • [3 года назад](http://e-maxx.ru/algo/dinic#comment-2149130565)

I have some doubts:  
1. What is the use of ptr[] array? It doesn't get updated anywhere, and always starts from 0.  
2. As far as I understand here, the f[u][v] matrix is not storing the residual flow from u to v, but actually the increased flow in the graph. But we are also storing negative of that flow in f[v][u]. But since this is not a residual network, it would become negative. How can we then reverse the flow, since we have a condition to check whether f[u][v]>0?

Аватар

**Paul** [Akash Goel](http://e-maxx.ru/algo/dinic#comment-2149130565) • [3 года назад](http://e-maxx.ru/algo/dinic#comment-2316010580)

1) Not always, if you run dfs the second time and bump into a node that was encountered by one of the previous dfs's then ptr[v] for that node will not be 0.  
2) If f[v][u] is negative then when you subtract it from c[v][u] you will have something positive, because double minus gives you plus.



Аватар

**vanvector** • [4 года назад](http://e-maxx.ru/algo/dinic#comment-1612133254)

В реализации алгоритма в функции bfs используется memset (d, -1, n \* sizeof d[0]);  
Далее идет сравнение d[i] с -1 в двух местах. Но здесь необходимо делать такое сравнение d[i] < 0 для проверки найдено ли расстояние от стока до i-ой вершины.

* + - •Аватар

**vanvector** [vanvector](http://e-maxx.ru/algo/dinic#comment-1612133254) • [4 года назад](http://e-maxx.ru/algo/dinic#comment-1612197166)

В реализации все верно. Все биты станут еDinitzами после memset, а -1 в знаковом типе представляется как 11...1.

[](https://disqus.com/by/sylapaliyew/)

[**Silap Aliyev**](https://disqus.com/by/sylapaliyew/) • [4 года назад](http://e-maxx.ru/algo/dinic#comment-1351438759)

Adjacency List Implementation seems to be incorrect. I sumbitted the code here - [informatics.mccme.ru/mod/st...](http://disq.us/url?url=http%3A%2F%2Finformatics.mccme.ru%2Fmod%2Fstatements%2Fview3.php%3Fid%3D262%26chapterid%3D2784%3AkcV2OBaMqn-uoUUM0bBSr4jS9HQ&cuid=1346507). It didn't pass some tests (WA). Can you please review your code?

* + [](https://disqus.com/by/sylapaliyew/)

[**Silap Aliyev**](https://disqus.com/by/sylapaliyew/) [Silap Aliyev](http://e-maxx.ru/algo/dinic#comment-1351438759) • [4 года назад](http://e-maxx.ru/algo/dinic#comment-1351451372)

Sorry, it is correct. I just tested another implementation and I recieved exactly the same results. It turned out, that, answer (maxflow) is exceeding INT\_MAX. :) Thanks anyway.

[](https://disqus.com/by/facebook-100000021053781/)

[**Igor Demidov**](https://disqus.com/by/facebook-100000021053781/) • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-855605395)

Почему без строчки код не работает на некоторых тестах (получает TL)? Казалось бы, мы ходим по-прежнему по слоистой сети, но разрешаем пихать поток нулевой стоимости. Вроде, это не влияет на асимптотику. Можете пояснить, пожалуйста? int dfs (int v, int flow) {  
if (!flow) return 0;

[Аватар](https://disqus.com/by/seland/)

[**seland**](https://disqus.com/by/seland/) [Igor Demidov](http://e-maxx.ru/algo/dinic#comment-855605395) • [4 года назад](http://e-maxx.ru/algo/dinic#comment-1382448909)

Потому что в этом случае разрешается ходить по ребрам с нулевой пропускной способностью, и, например лемма 2 уже не выполняется. Поэтому асимптотика может быть уже и не такая.

[](https://disqus.com/by/Puma_L/)

[**Puma\_L**](https://disqus.com/by/Puma_L/) • [6 лет назад](http://e-maxx.ru/algo/dinic#comment-722575937)

Что означают переменные n, s, t?

[Аватар](https://disqus.com/by/e_maxx/)

[**e\_maxx**](https://disqus.com/by/e_maxx/) **Модератор** [Puma\_L](http://e-maxx.ru/algo/dinic#comment-722575937) • [6 лет назад](http://e-maxx.ru/algo/dinic#comment-722582024)

Количество вершин, номер вершины-истока и номер вершины-стока.

Аватар

**AIFDu** • [6 лет назад](http://e-maxx.ru/algo/dinic#comment-651324394)

В поиске в глубину неверное условие while (qh < qt && d[t] == -1), возможно зацикливание.



[Аватар](https://disqus.com/by/e_maxx/)

[**e\_maxx**](https://disqus.com/by/e_maxx/) **Модератор** [AIFDu](http://e-maxx.ru/algo/dinic#comment-651324394) • [6 лет назад](http://e-maxx.ru/algo/dinic#comment-651341335)

А как будет зацикливание? Что-то я не понимаю. Тут написано обычное условие "qh < qt" (означающее "пока очередь не пуста") вместе с условием "d[t] == -1" (которое останавливает цикл раньше времени, если расстояние до t уже найдено).

[](https://disqus.com/by/disqus_bgr2KCjeMg/)

[**Borys**](https://disqus.com/by/disqus_bgr2KCjeMg/) • [6 лет назад](http://e-maxx.ru/algo/dinic#comment-593955666)

Пожалуйста, подскажите почему вы при добавлении ребра A - B с пропускной способность Cap делаете ребро A -> B (п.с. Cap) и B -> A (п.с. 0)? Речь про эту часть кода:

edge e1 = { a, b, cap, 0 };  
edge e2 = { b, a, 0, 0 };

Почему второе ребро B -> A п.с. 0 а не Cap? Если труба может пропускать 3 л/с в одну сторону, то что ей мешает пропускать столько-же в обратную?

[Аватар](https://disqus.com/by/e_maxx/)

[**e\_maxx**](https://disqus.com/by/e_maxx/) **Модератор** [Borys](http://e-maxx.ru/algo/dinic#comment-593955666) • [6 лет назад](http://e-maxx.ru/algo/dinic#comment-594070852)

Потому что эта функция добавляет ориентированное ребро: т.е. по нему поток может течь только в одну сторону (от A к B). Тем не менее, эта функция технически добавляет два ребра: само прямое ребро с его пропускной способностью, и обратное к нему - с пропускной способностью 0. Интуитивный смысл остаточного ребра - в том, что если алгоритм сначала пропустит по ребру некоторый поток, а потом "решит", что надо часть этого потока отменить - то он пропустит этот "излишний" поток по обратному ребру, тем самым отменив этот излишек на прямом ребре.

Если в сеть надо добавить неориентированное ребро (чтобы поток мог протекать в обе стороны), то это производится двумя вызовами функции add\_edge().

Аватар

**Lord\_F** [e\_maxx](http://e-maxx.ru/algo/dinic#comment-594070852) • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-793855357)

А разве в случае неориентированного ребра нельзя обойтись одним вызовом add\_edge() и в обе стороны сделать п.с. cap?

* + - * [1](https://disqus.com/embed/comments/?base=default&f=e-maxx&t_u=http%3A%2F%2Fe-maxx.ru%2Falgo%2Fdinic&t_d=MAXimal%20%3A%3A%20algo%20%3A%3A%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B8%D0%BD%D0%B8%D1%86%D0%B0%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0&t_t=MAXimal%20%3A%3A%20algo%20%3A%3A%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B8%D0%BD%D0%B8%D1%86%D0%B0%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0&s_o=default)

Аватар

**Lord\_F** [e\_maxx](http://e-maxx.ru/algo/dinic#comment-594070852) • [5 лет назад](http://e-maxx.ru/algo/dinic#comment-793889539)

А, да, нельзя, понял в чем тут загвоздка.